IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Joseph G. Laura	§	
		§	Group Art Unit: 2192
Serial No.:	10/723,967	§	
		§	Examiner: Wang, Ben C.
Filed:	November 26, 2003	§	
		§	Confirmation No. 9521
For:	APPLICATION MONITOR SYSTEM AND METHOD	§	
		§	
		§	

Mail Stop: AF
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

RESPONSE TO FINAL OFFICE ACTION

Commissioner:

Applicant acknowledges receipt of the Final Action dated April 16, 2008, and respectfully requests entry of the following amendments to the above-identified application. The changes made are shown by underlining the added text and striking through or double-bracketing the deleted text.

Claims are reflected in the listing of claims which begins on Page 2 of this paper.

Remarks/Arguments begin on Page 13 of this paper.

***Listing of the Claims*:**

1. (Currently Amended) A system for non-intrusively monitoring an application, comprising:

at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area;

a first module stored on a computer-readable medium that shares and attaches to [[a]] the shared memory area that is used by [[an]] the at least one application during real-time operation, the first module reads application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation;

a second module stored on a computer-readable medium in communication with the first module that requests the first module to read the application values, the second module[[-]] receives the application values from the first module; and

a third module stored on a computer-readable medium in communication with the second module[[-]] that displays the application values.

2. (Currently Amended) The system of Claim 1, wherein the shared memory area is further defined as a shared memory of the application.

3. (Currently Amended) The system of Claim 1, wherein the first module is further operable to attach to the shared memory area used by the at least one application to read the application values.

4. (Currently Amended) The system of Claim 1, wherein the application values are further defined as at least one application variable and a value for the at least one application variable.

5. (Original) The system of Claim 1, wherein the first module is further operable to communicate the application values to the second module in hypertext markup language format.

6. (Original) The system of Claim 1, wherein the third module is further defined as a graphical user interface.

7. (Currently Amended) The system of Claim 6, wherein the graphical user interface is further operable to receive an input identifying the application values to be read and operable to request the application values identified to the first module, via the second module, and wherein the first module is operable to read the requested application values data from the shared memory area and return the application variables to the graphical user interface, via the second module.

8. (Original) The system of Claim 6, wherein the graphical user interface is further operable to receive an input identifying requested application values to be displayed.

9. (Original) The system of Claim 1, wherein the first module is further operable as a socket server and wherein the second module is further operable as a socket client such that the first and second modules communicate via a socket connection.

10. (Currently Amended) The system of Claim 1, wherein the first module [[operable to]] reads application values stored in the shared memory area by the at least one application while the at least one application is running.

11. (Currently Amended) The system of Claim 10, wherein the first module [[operable to]] reads application values stored in the shared memory area by the at least one application without interfering with the operation of the at least one application.

12. (Currently Amended) A method of non-intrusively monitoring operation of an application, comprising:

running an application in a real-time manner;

creating a memory area;

generating, by the application, application values during operation of the application;

[[storing]] writing, by the application, the application values in [[a]] the memory area during the operation of the application;

reading, by a monitor, the memory area used by the application to obtain the application values, wherein at least one of the application values is not output by the application;

and

displaying the application values read from the memory area.

13. (Previously Presented) The method of Claim 12, further comprising:

requesting, by a client, application values from the monitor; and

communicating the application variables from the monitor to the client.

14. (Previously Presented) The method of Claim 13, further comprising:

requesting application values;

running a plurality of applications in a real-time manner;

generating application values stored in one or more memory areas during operation of the plurality of applications;

reading the one or more memory areas used by the plurality of applications to obtain the application values; and

displaying the requested application values.

15. (Currently Amended) The method of Claim 14, wherein the memory area is further defined as a block of [[shared]] memory and wherein the monitor reads [[the]] at least some of the application variables stored in the block of [[shared]] memory.

16. (Currently Amended) The method of Claim 13, further comprising providing a memory manager and wherein the monitor registers with the memory manager to obtain a location of the memory area used by the application to store the application values.

17. (Original) The method of Claim 13, further comprising:

generating new application values by the application stored in the memory area, at least one of the new application values defined as a new value for a variable of the application;

requesting, by the client, that the monitor re-read the application values stored in the memory area;

re-reading, by the monitor, the memory area to obtain the new application values.

18. (Currently Amended) The method of Claim 17, wherein the monitor reads the application values while the application is running.

19. (Original) The method of Claim 13, wherein the monitor is operable as a socket server and wherein the client is operable as a socket client such that the communication between the monitor and client is via a socket connection.

20. (Original) The method of Claim 12, wherein the application values are further defined as a variable of the application and a value of the variable.

21. (Currently Amended) A system for non-intrusively monitoring variables during operation of an application, comprising:

a compile listing stored on a computer-readable medium having an address map with an offset associated with each of a plurality of variables of an application; and

a module stored on a computer-readable medium that performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables of the application, the module performs attaching to an address space used by the application during real-time operation to obtain a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset.

22. (Previously Presented) The system of Claim 21, wherein the module is further operable to read the compile listing and convert at least one of the plurality of variables to the associated offset.

23. (Previously Presented) The system of Claim 21, wherein the module is further operable to search the compile listing and display the plurality of variables of the application for selection by a user.

24. (Original) The system of Claim 23, wherein the module is responsive to selection by the user of one of the plurality of variables to obtain the value for the selected one of the plurality of variables using the offset to locate the value of the variable in the address space.

25. (Original) The system of Claim 24, wherein the module is further operable to display the selected one of the plurality of variables.

26. (Original) The system of Claim 21, wherein the address space is further defined as a memory space and wherein the module attaches, using a socket layer, to the memory space used by the application.

27. (Original) The system of Claim 26, wherein the module attaches, using the offset, to the memory space used by the application via an operating system service.

28. (Previously Presented) The system of Claim 21, wherein the monitor is further operable, using the compile listing, to query the address map for one or more of the plurality of variables of the application.

29. (Original) The system of Claim 21, wherein the module is further defined as a subtask of the operating system.

30. (Previously Presented) The system of Claim 21, wherein the module is further operable to attach to the memory space where the application is operating and overwrite the value for one or more of the plurality of variables using the offset.

31. (Previously Presented) The system of Claim 21, wherein the module comprises:

a reader component operable to perform reading the compile listing and further operable to perform converting at least one of the plurality of variables of the application to the associated offset; and

a search component that performs receiving the associated offset of the at least one of the plurality of variables from the reader component, the search component operable to perform attaching to the application and further operable to locate the value of the at least one of the plurality of variables using the offset.

32. (Previously Presented) The system of Claim 21, further comprising a display component operably coupled to the module to perform receiving the value for the one or more of the plurality of variables, the display component operable to perform displaying the value.

33. (Original) The system of Claim 32, wherein the display component is operable to employ the value to display a heartbeat.

34. (Original) The system of Claim 32, wherein the display component is operable to employ the value to display as a percentage complete.

35. (Currently Amended) A system for non-intrusively monitoring COBOL application values, the system comprising:

~~a memory area;~~

a COBOL program stored on a computer-readable medium that creates a shared memory area through a technical layer, generates program values, and stores the program values in the shared memory area during real-time operation of the COBOL program; and

a COBOL monitor module stored on a computer-readable medium that shares the shared memory area with the COBOL program [[though]] through the [[a]] technical layer, and the COBOL monitor module reads the program values stored in the shared memory area by the COBOL program during real-time operation of the COBOL program.

36. (Currently Amended) The system of Claim 35, further comprising:

a second COBOL program operable to generate second program values and store the program values in the shared memory area during real-time operation of the second COBOL program, and wherein the COBOL monitor module is further operable to read the second program values stored in the shared memory area by the second COBOL program.

37. (Original) The system of Claim 35, further comprising:

a second memory area; and

a second COBOL program operable to generate second program values and store the program values in the second memory area during real-time operation of the second COBOL program, and wherein the COBOL monitor module is further operable to read the second program values stored in the second memory area by the second COBOL program.

38. (Original) The system of Claim 35, further comprising:

a user interface operable to monitor and display the application values; and

a client application in communication with the user interface and the COBOL monitor module, the client application operable to request the program variables of the COBOL program from the COBOL monitor module and provide the program variables to the user interface for display via the user interface responsive to a request from the user interface.

REMARKS

This application has been carefully considered in connection with the Final Office Action dated April 16, 2008. Reconsideration and allowance are respectfully requested in view of the following.

Summary of Rejections

Claims 1-34 were rejected under 35 USC § 103(a).

Claims 35-38 were rejected under 35 USC § 102(b).

Summary of Response

Claims 1-4, 7, 10-12, 15-16, 18, 21, and 35-36 are currently amended.

Claims 13-14, 22-23, 28, and 30-32 were previously presented

Claims 5-6, 8-9, 17, 19-20, 24-25, 26-27, 29, 33-34, and 37-38 remain as originally filed.

Summary of Claims Pending:

Claims 1-38 are currently pending following this response.

Response to Rejections

Sridharan, Tao-1, Tao-2 and Kashima do not disclose, teach or suggest non-intrusively monitoring a software application during real-time operation, whereby the application creates a shared memory area, stores application values in the shared memory area, and a monitor that shares the memory area reads the stored application values without interfering with the operation

of the software application involved. Generally, it is useful to monitor software (e.g., applications, programs, subroutines, modules, etc.) during normal operations. For example, conventional software applications may write intermediate information to certain internal variables, but the content of these intermediate variables is not output by the applications. Consequently, this content is not accessible to conventional software test or development programs without changing or encapsulating the software application involved. Therefore, it is more difficult for designers to develop, test or debug software applications using conventional techniques, without knowledge of the intermediate variables that were written as the software applications evolved. However, using one or more of the non-intrusive monitoring approaches disclosed in the pending application, a software application may be programmed to create a block of shared memory that can be shared by a software monitor, which can read application values that were written into the shared memory by the software application involved. Thus, for example, a developer may view an application's intermediate variables in order to understand and correct the operation of the application during its testing phase, or a business analyst may access the internal variables of multiple software applications in order to optimize the interactions between the multiple applications involved.

The pending application discloses systems and methods for non-intrusively monitoring a software application during real-time operation. The software application creates a shared memory area to be used for monitoring, and writes application values into the shared memory area while also performing allocated software tasks. A monitor module, which shares the memory area, reads the written application values while the software application is running in real-time. Thus, the monitor module attaches to the shared memory area during normal operation of the software application, which enables the monitor module to access all of the internal variables, intermediate and otherwise, that the software application writes to the shared memory area. Therefore, the

monitor module may access internal variable values of the software application that are otherwise not accessible by conventional testing or development techniques without changing or encapsulating the software application involved.

Sridharan is directed to a proposal for a non-intrusive framework for collecting performance statistics of Common Object Request Broker Architecture (CORBA)-based distributed systems. Sridharan describes an application of its non-intrusive framework to a large telecommunications system and the experimental results obtained. Specifically, Sridharan's study describes use of a Visibroker Object Request Broker (ORB) to deploy servers in the telecommunication system involved. The Visibroker ORB loads a performance instrumentation module together with a server into a single address space. According to Sridharan, sharing of the address space by an instance of the performance instrumentation module helps in collecting server specific information such as the CPU time, memory utilization, and the number of ORB threads launched. In other words, Sridharan generally describes a server and performance instrumentation module that share an address space so the performance instrumentation module can retrieve performance details for the particular server involved.

Tao-1 is directed to a monitoring framework that allows users to understand the memory behavior of parallel applications. Specifically, Tao-1 discloses use of a hardware monitor as part of an event-driven monitoring approach that provides a user with detailed information about low-level memory transactions in multiple applications. Based on the statistics of the observed memory transactions, data or threads can be redistributed or reallocated more rationally among processors in order to reduce remote memory accesses and improve parallel performance. Tao-1 generally describes certain shared memory programming models, such as Pthreads, Win32threads, and SPMD, which partition the shared data of an application into 4096 byte pages that are

distributed transparently among nodes. However, Tao-1 discloses no particular pertinent details about memory sharing or how it is performed.

Tao-2 is directed to a technique for visualizing the memory access behavior of shared memory applications on Non-Uniform Memory Access (NUMA) architectures. Specifically, Tao-2 discloses a visualization tool that displays monitored data in a user understandable way, in order to show the memory access behavior of shared memory applications. According to Tao-2, such tools require detailed information about the low-level memory transactions in the running program, and the only way to facilitate this without causing a high probe overhead is to perform the data acquisition using a hardware monitor capable of observing all memory transactions performed across the interconnection fabric. Also according to Tao-2, the disclosed hardware monitor traces memory references performed by the running program and gives the user a complete overview of the program's memory access behavior in the form of access histograms. However, similar to Tao-1, Tao-2 discloses no particular pertinent details about memory sharing or how it is performed.

Kashima is directed to an approach for constructing Web-based enterprise systems with distributed object technology. Kashima concentrates on issues to be considered while investigating the applicability of using distributed object technology in the Internet environment. Specifically, Kashima discusses the CORBA standard and maintains that with CORBA, connectivity with current systems is kept high because of CORBA's mainframe and COBOL support. Thus, Kashima teaches the use of CORBA for constructing the Web-based enterprise systems involved. Kashima mentions the use of unique name spaces in the Web environment, but Kashima does not disclose or suggest any details related to memory sharing or how it is performed.

As shown above, none of the above-cited art in the Final Office Action discloses, teaches or suggests non-intrusively monitoring a software application during real-time operation, whereby

the application creates a shared memory area, stores application values in the shared memory area, and a monitor that shares the memory area reads the stored application values without interfering with the normal operation of the software application, as claimed.

These distinctions, as well as others, will be discussed in greater detail in the analyses of the pending claims that follow.

Response to Rejections under 35 U.S.C. § 103

In the Final Office Action dated April 16, 2008, Claims 1-4, 6-11, 12-14, and 17-20 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan et al. (*On Building Non-Intrusive Performance Instrumentation Blocks for CORBA-based Distributed Systems,* March 2000, IEEE) ("Sridharan") in view of Tao et al. (*Understanding the Behavior of Shared Memory Applications Using the SMiLE Monitoring Framework,* March 2000, IEEE) ("Tao-1"). These rejections are respectfully traversed.

Claim 1:

I. Sridharan and Tao-1 do not disclose, teach or suggest at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area.

Claim 1 (as currently amended) recites, in part, "at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area."

The Final Office Action did not address these elements of claim 1, which were added by amendment in this response. Nevertheless, Applicant asserts that neither Sridharan nor Tao-1

disclose, teach or suggest the "at least one application" recited in claim 1. More precisely, neither Sridharan nor Tao-1 disclose, teach or suggest that "the at least one application creates a shared memory area and stores application values in the shared memory area" as claimed.

For example, Sridharan (on Page 3) describes its non-intrusive framework for collecting performance statistics of CORBA-based distributed systems as follows:

> The Visibroker ORB, which was used to deploy the servers, provided the facility to load the Performance Instrumentation (PI) module together with the server into a single address space while starting the server using the Java VM [3]. Also, the sharing of the address space by the instance of PI module helps in collecting server specific information such as CPU Time, memory utilization, and the number of ORB threads launched per server.
>
> Each instance of the PI module, called a PI instance, is loaded with every server. The PI instance retrieves performance details for that particular server.

As shown above, Sridharan describes its PI instance as sharing a single address space with a server to collect performance information for that server. Notably, however, Sridharan does not teach or suggest that either its PI instance or server includes an application to be monitored that creates the address space to be shared, and stores application values in the shared address space. In other words, Sridharan does not teach or suggest "at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area," as recited in claim 1.

Notably, Tao-1 does not cure the above-described deficiencies of Sridharan. For example, Tao-1 (Page 3) discusses shared memory in a description of the use of SCI Virtual Memory (SCI-VM) and cache memory in a simulation system of a SMiLE hardware monitor, as follows:

> SCI Virtual Memory (SCI-VM) [16], [8] is a concept for the implementation of a global virtual memory using hybrid hardware/software DSM. This concept can be applied in a fully

> transparent manner allowing the execution of shared memory applications on a global virtual memory without any changes. The memory is distributed at page granularity and remote pages are mapped from other nodes using SCI's HW-DSM mechanisms. The basic address entity for these mapping is the SCI physical address space comprising all physical memory within the cluster. From there, pages can be mapped into the PCI address spaces, and further into processes' virtual address spaces. There they are merged with the local pages forming a global virtual address space, the SCI Virtual Memory.
>
> On top of the SCI-VM several shared memory programming models including Pthreads, Win32threads, and SPMD [8] have been or are currently being implemented. In these models the shared data of an application is partitioned into 4096 bytes pages and usually distributed transparently among all nodes in a round-robin fashion. Some programming models allow an optional direct placement of some data structures onto specific nodes while the remainder of the data is allocated transparently.

As shown above, although Tao-1 generally describes the use of shared memory, Tao-1 does not teach or suggest any relevant details in that regard. Thus, Tao-1 does not disclose, teach or suggest "at least one application stored on a computer-readable medium, the at least one application creates a shared memory area and stores application values in the shared memory area," as recited in claim 1.

II. Sridharan and Tao-1 also do not disclose, teach or suggest a first module that reads application values from the shared memory that have been stored by the application during real-time operation.

Claim 1 also requires, "a first module ... that shares and attaches to the shared memory area that is used by the at least one application during real-time operation, the first module reads application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation."

The Final Office Action relied on the disclosure in Sridharan of its PI module to read on these elements of claim 1. As established in the summary section above, Sridharan discloses techniques for non-intrusively monitoring the performance of various servers at run-time. For example, Sridharan discloses in Paragraph 2 of Page 3, "Each instance of the PI module, called a PI instance, is loaded with every server. The PI instance retrieves performance details for that particular server." As such, Sridharan merely teaches that its PI instance is used to retrieve performance information for the server involved. As such, Sridharan does not teach or suggest that "the first module reads application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation," as claimed. Notably, Sridharan discloses in the last Paragraph of Page 3, "A GUI was implemented to display the various performance data (e.g., average, maximum, minimum latency values for methods in various servers)." Accordingly, Applicant respectfully submits that a PI instance that obtains performance measurements for methods operating on a server is clearly different from a module reading "application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation," as claimed.

Furthermore, claim 1 also requires "a first module ... that shares and attaches to the memory area that is used by the at least one application." Thus, if the Final Office Action interprets Sridharan such that its PI module represents the "first module" recited in claim 1, it follows then that Sridharan's server would represent the "at least one application" also recited in this claim. As such, Sridharan discloses in the first paragraph of Page 3 that the PI module and the server are loaded into a single address space. However, Sridharan also discloses in the third paragraph of Page 3 that "P(P1, P2)" shown in Figure 4 represents parameter values P1 and P2 in an incoming request for a service P, which is passed by the PI instance to the server F. Notably,

the parameter values, P1 and P2, are not stored in the single address space by the server. Therefore, the parameter values, P1 and P2, do not teach or suggest the "application values ... that have been stored in the shared memory area by the at least one application," as required by claim 1. (Underlining added for emphasis).

Tao-1 does not cure the above-described deficiencies of Sridharan in this regard. For example, Tao-1 discloses in its section IV on Page 5, that a visualizer may be used by a programmer before execution to rationally distribute data among processors in a distributed shared memory (DSM) system. According to Tao-1, data locality can be improved by the programmer and the system "before an execution, i.e., the data is already placed in an appropriate processor at allocation time". Tao-1 also states, "In the case when the programmer is responsible for the locality optimization a visualizer is needed to show the monitoring information to the programmer vividly." Tao-1 states further, "A mechanism for mapping each memory location observed by the monitoring system to its corresponding program data structure identifier (procedure and variable names) is being implemented as well." As shown by the above, the techniques disclosed in Tao-1 aid a programmer in optimizing the distribution of data among processors before execution. However, Tao-1 does not teach or suggest reading "application values" as claimed. Tao-1 merely discloses the displaying of application data structure identifiers, such as procedures and variable names. Furthermore, Tao-1 teaches displaying of the application data structure identifiers before execution, but not "during real-time operation," as claimed. Therefore, Tao-1 does not disclose, teach or suggest reading "application values from the shared memory area that have been stored in the shared memory area by the at least one application during real-time operation," as recited in claim 1.

For at least the reasons established above in sections I and II, Applicant respectfully submits that independent claim 1 is not obvious over Sridharan in view of Tao-1 and respectfully requests allowance of this claim.

Claims Depending from Claim 1:

Claims 2-4 and 6-11 were also rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-1. These rejections are respectfully traversed.

Claim 5 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-1 and further in view of Hiroshi Kashima *(An Approach for Constructing Web Enterprise Systems on Distributed Objects*, Jan., 2000, IBM) ("Kashima"). This rejection is respectfully traversed.

Dependent claims 2-11 depend directly or indirectly from independent claim 1 and incorporate all of the limitations thereof. Accordingly, for at least the reasons established in sections I and II above, Applicant respectfully submits that claims 2-11 are also not obvious over Sridharan in view of Tao-1 and respectfully requests allowance of these claims. Applicant respectfully submits that Kashima does not cure the deficiencies of Sridharan and Tao-1 as noted above.

Claim 12:

III. Neither Sridharan nor Tao-1 disclose, teach or suggest writing, by the application, the application values in a memory area during the operation of the application.

Claim 12 recites, in part, "writing, by the application, the application values in a memory area during the operation of the application."

The Final Office Action addressed these elements of claim 12 on Page 9 with respect to Sridharan, but the Final Office Action did not cite just where in Sridharan (or Tao-1) these elements are disclosed. Nevertheless, Applicant asserts that neither Sridharan nor Tao-1 disclose, teach or suggest the above-described "writing, by the application..." features recited in claim 12.

Specifically, as described above with respect to the rejection of claim 1, Sridharan (on Page 3) describes its non-intrusive framework for collecting performance statistics of CORBA-based distributed systems as follows:

> The Visibroker ORB, which was used to deploy the servers, provided the facility to load the Performance Instrumentation (PI) module together with the server into a single address space while starting the server using the Java VM [3]. Also, the sharing of the address space by the instance of PI module helps in collecting server specific information such as CPU Time, memory utilization, and the number of ORB threads launched per server.
>
> Each instance of the PI module, called a PI instance, is loaded with every server. The PI instance retrieves performance details for that particular server.

As shown above, Sridharan describes its PI instance as sharing a single address space with a server to collect performance information for that server. Notably, however, Sridharan does not teach or suggest that either its PI instance or server is or includes an application that writes application values in the single address space. In other words, Sridharan does not teach or suggest "writing, by the application, the application values in a memory area during the operation of the application," as claimed. Also, Tao-1 does not cure the deficiencies of Sridharan in that regard.

IV. Neither Sridharan nor Tao-1 disclose, teach or suggest a monitor that reads at least one application value that is not output by the application.

Claim 12 also recites, in part, "reading, by a monitor, the memory area used by the application to obtain the application values, wherein at least one of the application values is not output by the application." In other words, the subject matter of claim 12 provides a method of accessing internal variable values of an application that were otherwise not accessible by conventional methods.

As established above in section II (with respect to claim 1), neither Sridharan nor Tao-1 teach or suggest reading application values from memory. Therefore, it follows logically that neither Sridharan nor Tao-1 also do not teach or suggest that "at least one of the application values is not output by the application," which is required by claim 12. A detailed reading of Sridharan and Tao-1 support that fact. Therefore, for at least these reasons and the reasons established above in section III, Applicant respectfully submits that independent claim 12 is not obvious over Sridharan in view of Tao-1 and respectfully requests allowance of this claim.

Claims Depending from Claim 12:

Dependent claims 13-14 and 17-20 were also rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-1. These rejections are respectfully traversed.

Claim 15 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-1 and further in view of Kashima. This rejection is respectfully traversed.

Claim 16 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-1 and further in view of Huang et al., *(Operating System Support for Flexible Coherence in Distributed Shared Memory,* 1996, IEEE (hereinafter, "Huang"). This rejection is respectfully traversed.

Dependent claims 13-20 depend directly or indirectly from independent claim 12 and incorporate all of the limitations thereof. Accordingly, for at least the reasons established in

sections III and IV above, Applicant respectfully submits that claims 13-20 are not obvious over Sridharan in view of Tao-1 and respectfully requests allowance of these claims. Applicant respectfully submits that Kashima and Huang do not cure the deficiencies of Sridharan and Tao-1 as noted above.

Claim 21:

Claim 21 was rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan et al. in view of Jie Tao et al. *(Visualizing the Memory Access Behavior of Shared Memory Applications on NUMA Architectures,* Springer-Verlag Berlin Heidelberg 2001, pp. 861-870) ("Tao-2"). This rejection is respectfully traversed.

V. Sridharan and Tao-2 do not disclose, teach or suggest a compile listing having an address map with an offset associated with each of a plurality of variables of the application.

Claim 21 recites, in part, "a compile listing stored on a computer-readable medium having an address map with an offset associated with each of a plurality of variables of the application."

The Final Office Action relied on the following disclosure in section 3.4 of Tao-2 to read on these elements of claim 21:

> [T]he visualizer offers a 'Data structure' window to reflect this mapping. Instead of displaying only one page or section of interest as it is the case in the 'Node diagram' and the 'Page analysis' windows, this window shows all the shared variables occurring in a source code. In combination with the 'Access histogram', the 'Data structure' window provides users with a global overview of accesses to the complete working set of the application.

Notwithstanding the assertions made in the Final Office Action, as shown above, this section of Tao-2 clearly does not disclose, teach or suggest use of a "compile listing" as claimed. Tao-2 merely describes a data structure "window" that shows shared variables occurring in a

source code. In fact, Tao-2 appears to avoid the use of compile listings with the use of its data structure "window," when Tao-2 states later in section 3.4 that "many troubles and inaccuracies related to modifying compilers or extracting symbol information from binaries are avoided."

Furthermore, nowhere in the above-cited section of Tao-2 (or anywhere else in Tao-2) is there any mention or suggestion of a "compile listing ...having an address map with an offset associated with each of a plurality of variables of an application," as required by claim 21. Also, Sridharan does not cure the deficiencies of Tao-2 in this regard.

VI. Sridharan and Tao-2 also do not disclose, teach or suggest a module that attaches to the address space used by the application during real-time operation to obtain a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset.

Claim 21 also recites, in part, "a module stored on a computer-readable medium that performs reading of the compile listing and obtaining the offset of at least one of the plurality of variables of the application, the module performs attaching to an address space used by the application during real-time operation to obtain a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset."

The Final Office Action also relied on the above-cited portions of Tao-2 to read on these elements of claim 21. Nevertheless, as shown above, contrary to the assertions made in the Final Office Action in this regard, Tao-2's description of its "Data structure" window clearly does not disclose, teach or suggest the "module" recited in claim 21. Moreover, Tao-2 does not disclose, teach or suggest use of a module that obtains an offset of at least one variable of the application, that attaches to an address space used by the application during real-time operation of the

application, or that obtains a value for one or more of the plurality of variables written to the address space by the application during the real-time operation of the application using the offset, as claimed. Also, Sridharan does not cure the deficiencies of Tao-2 in this regard. Therefore, for at least the reasons established above in sections V and VI, Applicant respectfully submits that independent claim 21 is not obvious over Sridharan in view of Tao-2 and respectfully requests allowance of this claim.

Claims Depending from Claim 21:

Claims 23, 26, 28, and 32 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-2. These rejections are respectfully traversed.

Claims 22, 24-25, 27 and 29-31 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Sridharan in view of Tao-2 and further in view of Huang. These rejections are respectfully traversed.

Claims 33-34 were rejected under 37 U.S.C. 103(a) as being unpatentable over Sridharan in view of Tao-2 and further in view of Tao-1. These rejections are respectfully traversed.

Dependent claims 22-34 depend directly or indirectly from independent claim 21 and incorporate all of the limitations thereof. Accordingly, for at least the reasons established in sections V and VI above, Applicant respectfully submits that claims 22-34 are not obvious over Sridharan in view of Tao-2 and respectfully requests allowance of these claims. Applicant respectfully submits that Huang and Tao-1 do not cure the deficiencies of Sridharan and Tao-1 as noted above.

Response to Rejections under 35 U.S.C. § 102

Claim 35:

Claim 35 was rejected under 35 U.S.C. § 102(b) as being anticipated by Kashima. This rejection is respectfully traversed.

VII. Kashima does not disclose a COBOL program that creates a shared memory area through a technical layer.

Claim 35 (as currently amended) recites, in part, "a COBOL program stored on a computer-readable medium that creates a shared memory area through a technical layer."

The Final Office Action did not address these elements of claim 35, which were added by amendment in this response. Nevertheless, Applicant asserts that Kashima does not disclose the "COBOL program" recited in claim 35. More precisely, as explained in Paragraph [0026] of the pending application, COBOL programs, unfortunately, do not provide support for shared memory. Thus, some embodiments of the pending application include a technical layer, which enables COBOL programs to avail themselves of shared memory. Such a technical layer provides, among other things, a shared memory routine that provides a COBOL program with the functionality to enable memory sharing on a computer between applications, such as, for example, memory sharing between the COBOL program and COBOL monitor module recited in claim 35. Notably, Kashima does not disclose or suggest the use of a technical layer, or memory sharing through use of such a technical layer, between its COBOL applications, as claimed.

For example, the Final Office Action relied on the following disclosure in the first paragraph of section 2-2 in Kashima to read on the "COBOL program" and "technical layer" recited in claim 35:

> At execution time, a function called ORB (Object Request Broker) plays a central role as a software bus for services such as server detection and request transfer. It makes it possible for clients to access distributed objects in the same way as through local access. In the server side, there is a function called BOA (Basic Object Adapter) which does the initiation and control of distributed objects.

Notably, as shown above, Kashima does not disclose "a COBOL program ...that creates a shared memory area through a technical layer," as claimed. For example, the ORB described in Kashima merely functions as a software bus. As such, the ORB described in Kashima does not function as a technical layer to facilitate the creation of a shared memory area by a COBOL program or any other type of application. Consequently, contrary to the assertions made in the Final Office Action, Kashima does not disclose "a COBOL program stored on a computer-readable medium that creates a shared memory area through a technical layer,' as recited in claim 35.

VIII. Kashima also does not disclose a COBOL monitor module that shares the shared memory area with the COBOL program through the technical layer.

Claim 35 (as currently amended) also recites, in part, "a COBOL monitor module stored on a computer-readable medium that shares the shared memory area with the COBOL program through the technical layer."

For example, the Final Office Action relied on the following disclosure in the third paragraph of section 1-4 in Kashima to read on the "COBOL monitor module" recited in claim 35:

"In the case of CORBA, the connectivity with current systems is kept high because of its mainframe and COBOL support." As can be seen, this sentence merely mentions that CORBA provides COBOL support, but it clearly does not describe "a COBOL monitor module stored on a computer-readable medium that shares the shared memory area with the COBOL program through the technical layer," as claimed.

For at least the reasons established above in sections VII and VIII, Applicant respectfully submits that independent claim 35 is not anticipated by Kashima and respectfully requests allowance of this claim.

Claims Depending from Claim 35:

Claim 36-38 were also rejected under 35 U.S.C. § 102(b) as being anticipated by Kashima.

Dependent claims 36-38 dependent directly or indirectly from independent claim 35 and incorporate all of the limitations thereof. Accordingly, for at least the reasons established in sections VII and VIII above, Applicant respectfully submits that claims 35-38 are not anticipated by Kashima and respectfully requests allowance of these claims.

CONCLUSION

Applicant respectfully submits that the present application is in condition for allowance for the reasons stated above. If the Examiner has any questions or comments or otherwise feels it would be helpful in expediting the application, the Examiner is encouraged to telephone the undersigned at (972) 731-2288.

The Commissioner is hereby authorized to charge payment of any further fees associated with any of the foregoing papers submitted herewith, or to credit any overpayment thereof, to Deposit Account No. 21-0765, Sprint.

Respectfully submitted,

Date: June 16, 2008

/Michael W. Piper/
Michael W. Piper
Reg. No. 39,800

Conley Rose, P.C.
5601 Granite Parkway, Suite 750
Plano, Texas 75024
(972) 731-2288
(972) 731-2289 (facsimile)

ATTORNEY FOR APPLICANT